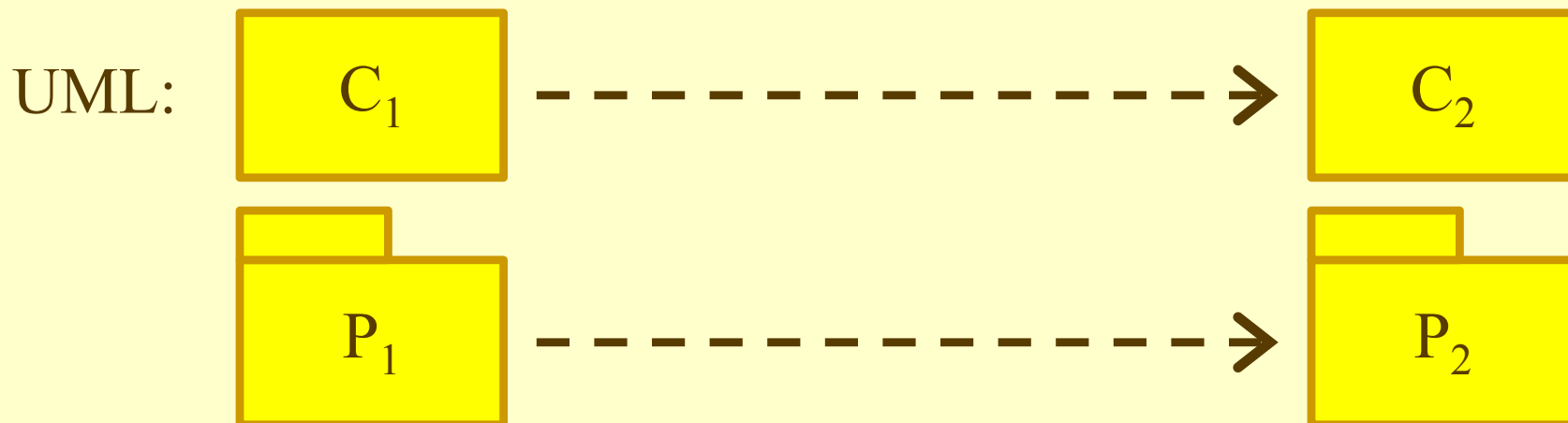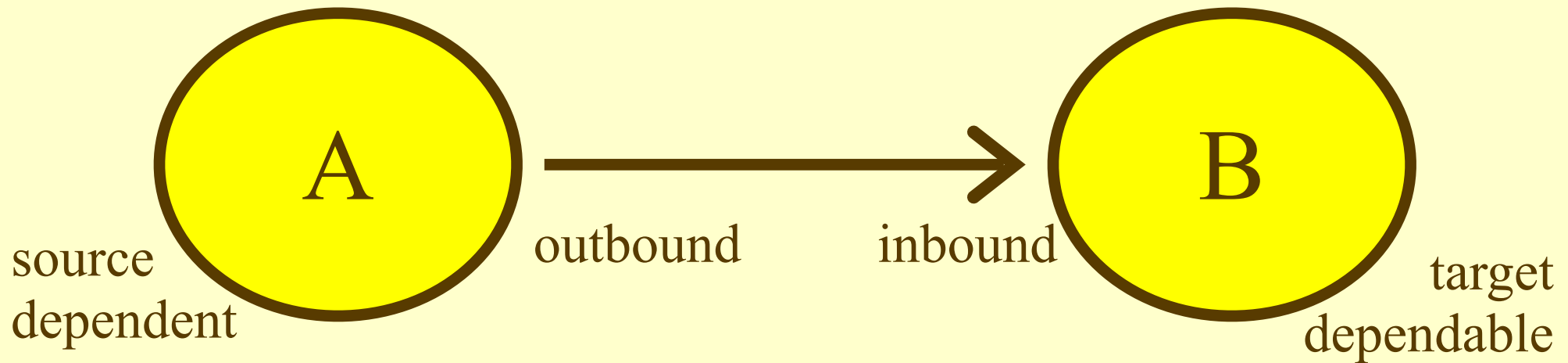# Tutorial
for Dependency Finder

by Jean Tessier,
March 30, 2004

What are dependencies?

Functioning of A requires the presence of B.

A

source
dependent

outbound

inbound

B

target
dependable

UML: C_1 - - - - - - - - - ▸ C_2
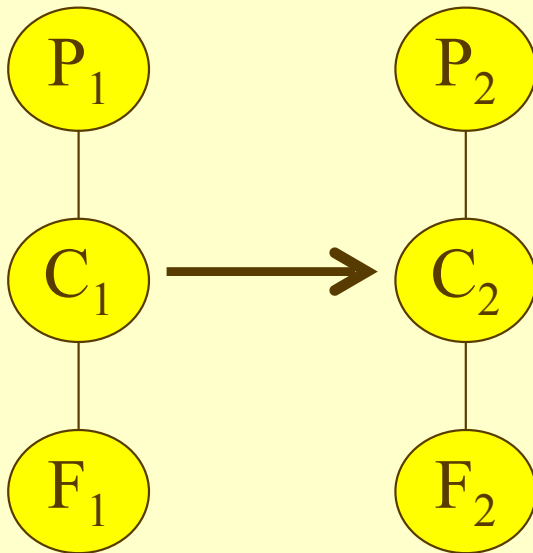
P_1 - - - - - - - - - ▸ P_2

Why Should I Care?

- Improve compilation time
  - Smaller parts are easier on the compiler

- Verify encapsulation
  - Validate design assumptions

- Indicator of complexity

- Assist with maintenance
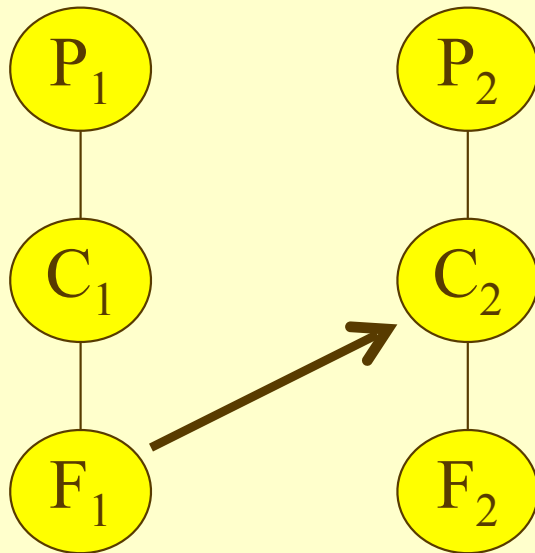 - Impact of change

Where Do They Come From?

P_1 P_2

$C_1 \longrightarrow C_2$

F_1 F_2

- Class-to-Class
  - Class inheritance via **extends** keyword
  - Interface realization via **implements** keyword
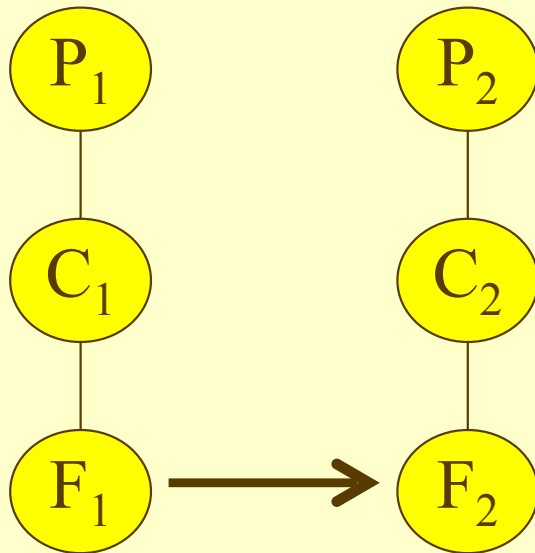 - From fields and methods of the class

Where Do They Come From?

P_1

P_2

C_1

C_2

F_1

F_2

- Feature-to-Class
  - Field type
  - Method parameter type
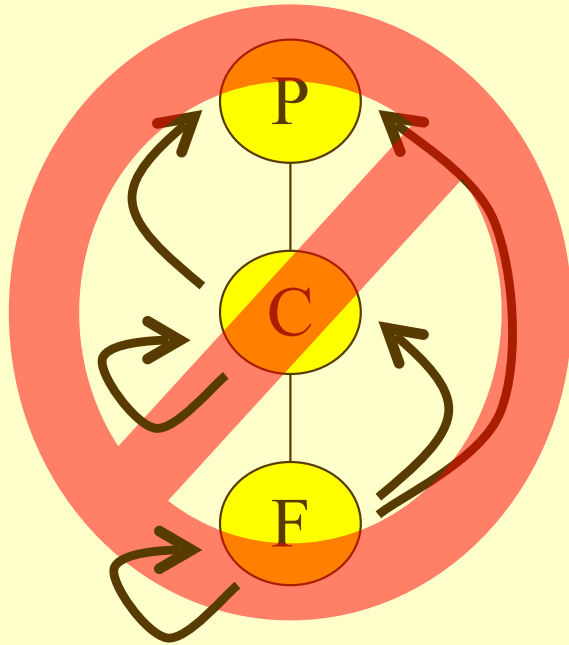 - Exception types in **throws** clause
  - Local variable type

5

Where Do They Come From?

P_1

P_2

C_1

C_2

F_1 → F_2

- Feature-to-Feature
  - Field access
  - Method call
  - Method cannot depend on itself, as in recursive calls

No Dependency on Self

- Does not add information
- Compile time
  - Class is unit of compilation
- Runtime
  - Class is unit of loading
- Maintenance
  - Already editing class or feature

Explicit Dependencies

```
P₁        P₂

C₁ ──────→ C₂

F₁ ──────→ F₂
```
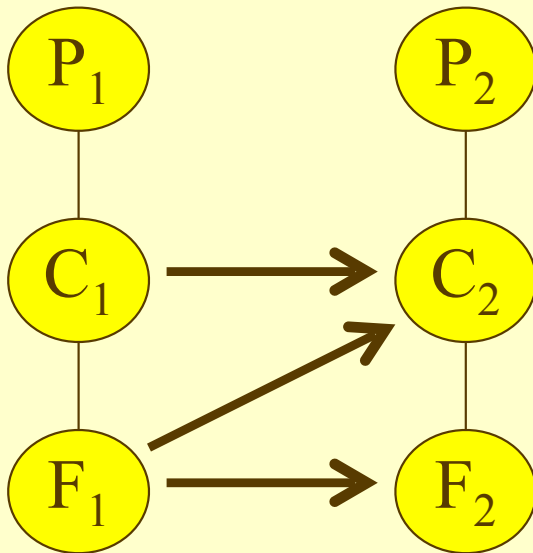
- Source code
  - **`static final`** constants
  - Needed for compilation

- Compiled code
  - Fully qualified names
  - No 3rd party JARs
  - Needed for runtime

Implicit Dependencies
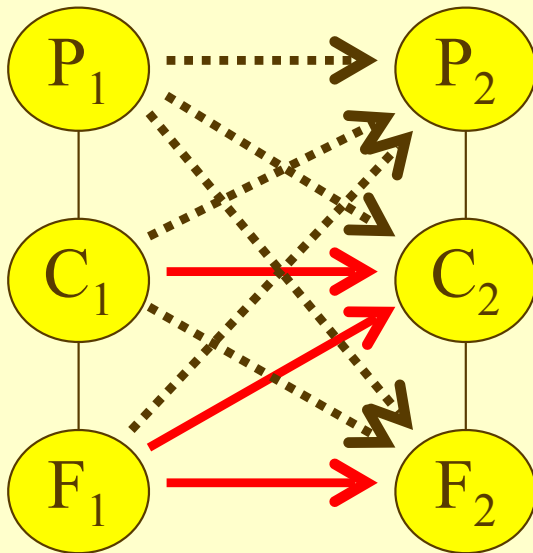
- Derived from explicit
- Minimize
  - Reduce space
  - Augment time
- Maximize
  - Augment space
  - Reduce time

Implicit Dependencies

classes of
a package
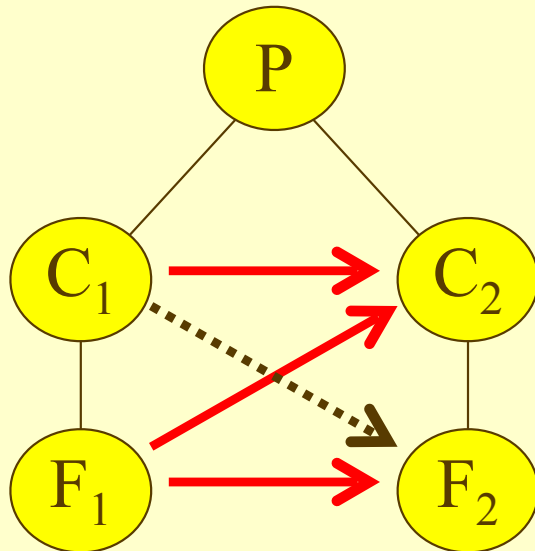
features of
a class
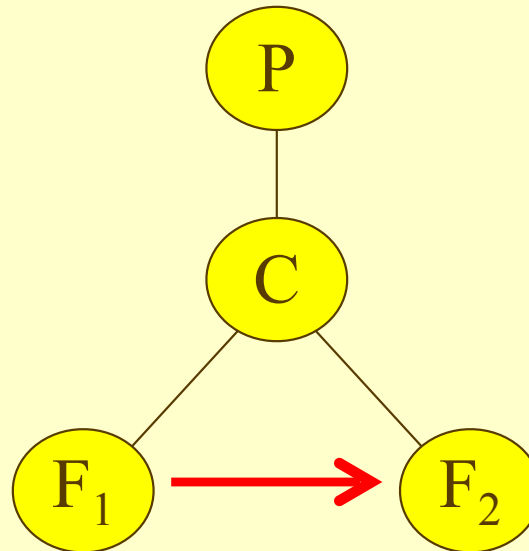
self
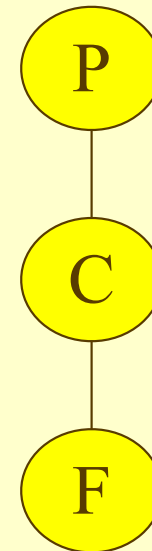
Package Instability (from Robert C. Martin)

C_a Number of classes outside the package that depend upon classes inside the package.

C_e Number of classes outside the package that classes inside the package depend upon.

$$I = \frac{C_e}{C_a + C_e}$$

- High C_e or low C_a means hard to change
- If C_a and C_e are 0, then $I = 0$
- Depend on packages with lower *I* than you

A vs I Graph (from Robert C. Martin)

1

uselessness

$A + I - 1 = 0$

A

pain

0   I   1
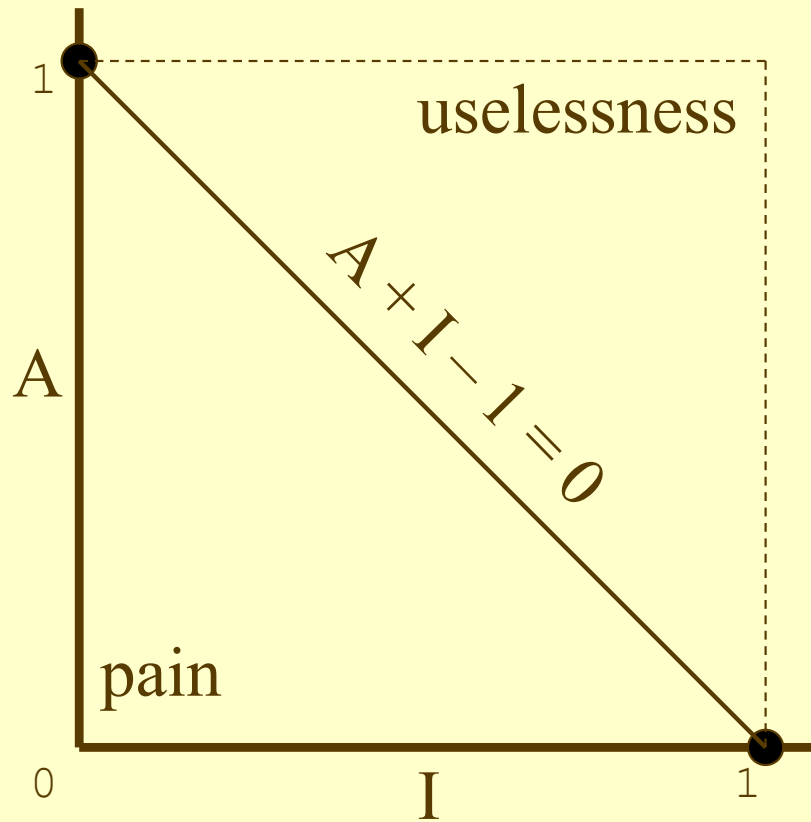
- Depend on abstract packages
- Concrete packages should have no dependents

Are Dependencies Transitive?
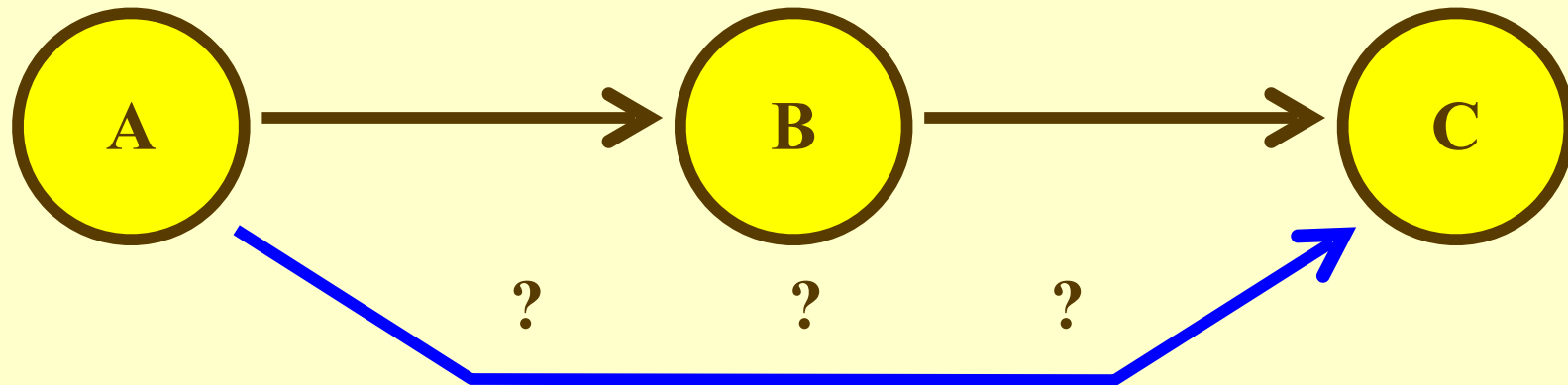
- It depends! ☺
- Case-by-case, not automatic
- Is part of B that A depends on impacted by changes in C?

Transitive Closure

callers
dependents

dependencies

Transitive Closure

Package-to-Package Example

Default configuration

Complete package-level graph

Package-to-Package Example

Excluding `/java/`

Don't list dependencies on anything with `java`

Why Do I See `java <-- jeantessier?`

Scope Factory

Filter Factory
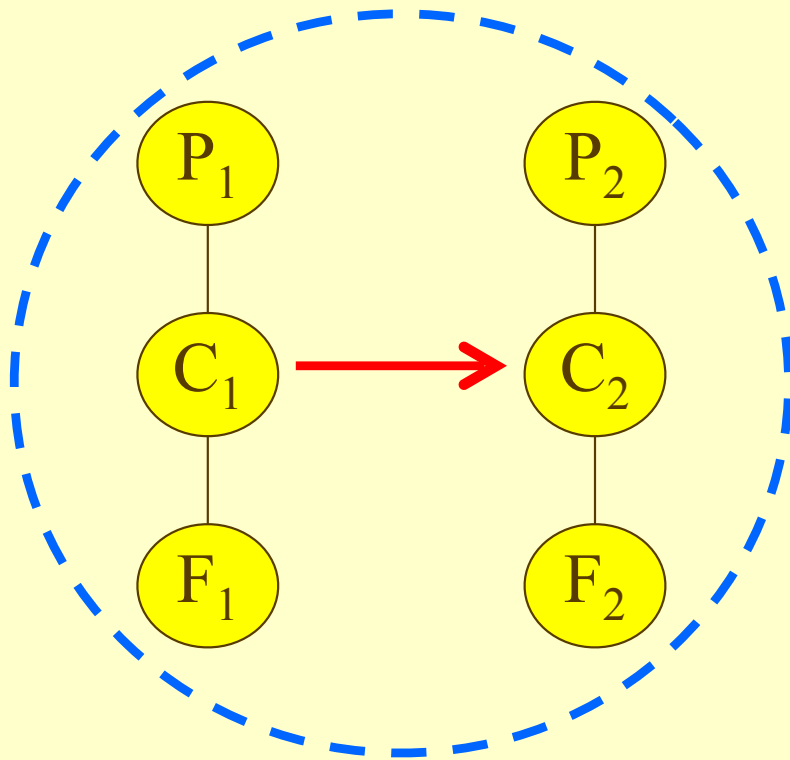
original

copy

Package-to-Package Example

Excluding **/^java/**

Don't list dependencies on **java.***

Package-to-Package Example

Including **/^com.jean/**

Only list dependencies with **com.jean***

Package-to-Package Example

Including **/^com.jean/, /^test/**

Only list with **com.jean*** and **test.***

Package-to-Package Example

Excluding `/^java/`

Don't list packages in `java.*`

Package-to-Package Example

Including `/com.jean/, /^test/`

Only list `test.*` and `com.jean*`

Feature-to-Feature Example

Including **/Node.Accept/**

Calls to ***Node.Accept()** methods

Feature-to-Feature Example

Select programming elements

☐ packages ☐ classes ☑ features

including: `/Node.Accept\(/` excluding:

Show dependencies

☐ packages ☐ classes ☑ features

including: `//` excluding: `/Test/`

Results

Dependencies Closure Metrics

Show: ☑ inbounds ☐ outbounds ☐ empty nodes

Select programming elements

☐ package ☐ class ☑ feature

including: `/Node.Accept\(/` excluding:

Show dependencies

☐ package ☐ class ☑ feature

including: `//` excluding: `/Test/`

Show dependencies ☑ to element ☐ from element ☐ (empty elements)

Including `/Node.Accept\(/`

Calls to `*Node.Accept()` methods, less output

Feature-to-Feature Example

Including `/(Node)\.\1\(/`

Calls to constructors of `Node`

Feature-to-Feature Example

Including `/(\w*Node)\.\1\(/`

Calls to constructors of classes ending in `Node`

Feature-to-Feature Example

Including **/NodeFactory/**

Calls to methods of **NodeFactory**

Transitive Closure Example

Shows `Accept*() methods`

Transitive Closure Example

Select programming elements		
☐ packages ☐ classes ☑ features		
including:	excluding:	
/Node.Accept/		

Show dependencies		
☐ packages ☐ classes ☑ features		
including:	excluding:	
/^com.jean/		

Results

| Dependencies | Closure | Metrics |

Follow inbounds: 1   Follow outbounds: 0

Start with programming elements

☐ package ☐ class ☑ feature

including: /Node.Accept/

excluding:

Follow dependencies

☐ package ☐ class ☑ feature

including: /^com.jean/

excluding:

Follow inbounds: 1   Follow outbounds: 0

Direct callers of `Accept*()`

Transitive Closure Example

Select programming elements
☐ packages ☐ classes ☑ features
including: /Node.Accept/
excluding:

Show dependencies
☐ packages ☐ classes ☑ features
including: /^com.jean/
excluding:

Results
Dependencies | Closure | Metrics
Follow inbounds: 2 Follow outbounds: 0

Start with programming elements
☐ package ☐ class ☑ feature
including: /Node.Accept/
excluding:

Follow dependencies
☐ package ☐ class ☑ feature
including: /^com.jean/
excluding:

Follow inbounds: 2 Follow outbounds: 0

Second degree callers of `Accept*()`

Transitive Closure Example

Everything reachable from **Accept*()**

Perl Regular Expressions

- `[A-Z]` Capitals
- `\w` Alphanumeric `[A-Za-z0-9_]`
- `\W` Non-alpha `[^A-Za-z0-9_]`
- `\w+` At least one alphanumeric
- `\s` Space, including tab and eol
- `\s*` Zero or more white space
- `\.` '.' as opposed to any character

The End

depfind.sourceforge.net